Homework #2 (10 points)

This document is set to allow comments. If anything is not clear, please just add a comment.

Purpose

The purpose of this assignment is to help you demonstrate your understanding of software weaknesses in web applications. This assignment will help prepare you for finding and fixing vulnerabilities in Web applications written using Python.

How to Submit it

Upload your deliverables on Gradescope.
You can upload the contents of a folder by dragging-and-dropping them into Gradescope. Alternatively, you can zip all the files and upload the zip file (Gradescope will automatically extract the files upon submission).
If there is any large file in the submission, gradescope throws an error message. Please contact the instructor.

Deliverables

- A PDF file with your answers to the conceptual questions in Part 1 and Part 2
- The source code for the website with the fixes you implemented in Part 3
    - If you're using pipenv/virtualenv, make sure to ***not*** include the hidden folders with all the binaries for Python libraries (that would make the zip file too large for Gradescope to accept).

Task

Machine Requirements:

- This homework requires that you have Python and Django installed on your machine. *(The code has been developed using Django v3.1.13. It should work just fine on v4.0 too)*.
- Running the code:
    1. Clone the course repository
    2. "cd" into the folder that has the source code for the Django web application
       `cd HW2/website`
    3. Run the web application using:
       `python manage.py runserver` (in your machine, it might be python3.x)
    4. Access http://localhost:8000/tasktracker/ in your web browser
        a. It will redirect you to a login page. There are three accounts created:
           Username: `user1` / **Password**: `p*2sfHQP==`
           Username: `user2` / **Password**: `V?vZT+MD4e`

If there are no issues with your environment, you'd be able to access the task tracker app.

Part 1: Conceptual Questions

- **Q1:** Read Django's documentation and answer the following question:
    - What is the purpose of `{% csrf_token %}` in the templates files?
        - The first defense against CSRF attacks is to use a side effect free request such as GET requests. Unsafe methods include POST, PUT and DELETE.
        - The token tag is used for CSRF protection. In any template that uses a POST form, it is safe practice to use the `{% csrf_token %}` tag IF the form is for an internal URL. (This tag should not be used for POST forms that target external URLs as this would cause the CSRF token to be linked.)
- **Q2:** What could happen if a developer forgets to add the `{% csrf_token %}` tag on their template that uses an HTML form?
  Django might not set the CSRF token cookie.
  This commonly happens when forms are dynamically added to the page.
  Django mitigates this by providing a view decorator which forces the setting of this cookie [`ensure_csrf_cookie()`].
- **Q3:** What is the CWE ID associated with the security problem that arises with forgetting to use `{% csrf_token %}`?
  CWE-352

Part 2: Finding Issues

In this homework part, you will play the role of a *software consultant* hired to identify the security vulnerabilities in a Web application. This is a simple Web application where a user can add/delete/lists tasks (i.e., a task management app). In light of that, answer the following question:

- **Q1:** Enumerate all the vulnerabilities you found in the web application. Include in your answer the following:
  #1: Broken Access Control
    - Where they are located in the code
      No check to see if the user is properly logged in in views.py delete function

```
def delete(request, pk):
    # uses ORM to delete the task
    task = Task.objects.get(id = pk)
    task.delete()
    # redirects user to index page
    return HttpResponseRedirect(reverse(f'tasktracker:index'))
```

- How to exploit it
  - User can logout and then delete a task (while logged out for the previous user) http://127.0.0.1:8000/tasktracker/delete/16 would delete task 16

- The corresponding CWE-ID (i.e., vulnerability type): CWE-284

#2: Bypass via Directory Traversal

- Where they are located in the code

  Delete function in view.py

```
def delete(request, pk):
    # uses ORM to delete the task
    task = Task.objects.get(id = pk)
    task.delete()
    # redirects user to index page
    return HttpResponseRedirect(reverse(f'tasktracker:index'))
```

- How to exploit it
  - User can delete a task from the log in page via directory traversal

- The corresponding CWE-ID (i.e., vulnerability type): CWE-22

#3: Incremental id - Insecure Design

- Where they are located in the code
  - ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
- How to exploit it
  - An AutoField is an IntegerField that automatically increments according to available IDs. One usually won't need to use this directly because a primary key field will automatically be added to your model if you don't specify otherwise.
- The corresponding CWE-ID (i.e., vulnerability type): CWE-342

#4: Security Misconfiguration

- Where they are located in the code
  - settings.py

```
# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []
```

- How to exploit it
  - If a security feature is not disabled it will not provide protection. In this case, debug should not be on during production.
  - Attacker can learn more about your program from the error page that will display on a 404.
- The corresponding CWE-ID (i.e., vulnerability type): CWE-388

#5: Security Key

- Where they are located in the code

- settings.py

```
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '#n6kks0cs$a-7k*67)k(nof$7z6&l+u97ea)nl_r8frg_mrmd1'
```

  - How to exploit it
    - Use of a hard-coded key significantly increases the chance that encrypted data might be recovered.
  - The corresponding CWE-ID (i.e., vulnerability type): CWE-321

<u>Extras that I was unable to determine how to fix.</u>

  - Session Fixation
    - Where they are located in the code
      - **I could not find a logout function. I assume this is part of the problem. The user is not completely being logged out?**
    - How to exploit it
      - If the user logs out and leaves the browser open, another user can retreat to the previous page and be logged in and have access to viewing the tasks without having to log in again.
    - The corresponding CWE-ID (i.e., vulnerability type): CWE-384
  - Lack of Lock Out Mechanism
    - Where they are located in the code
      - Unsure how to determine the number of log in attempts
    - How to exploit it
      - User could try to brute force crack the password by attempting logins with a script.
    - The corresponding CWE-ID (i.e., vulnerability type): CWE-307

Part 3: Fixing Issues

Now you will play the role of the *software developer* that will fix the identified security vulnerabilities.

- **Q1:** Change the code provided to fix all the vulnerabilities you found in the web application.

  #1: Broken Access Control &

  #2: Bypass via Directory Traversal

```
# Deletes a task (based on its primary key) and redirect it back t
def delete(request, pk):
    if request.user.is_authenticated:
        # uses ORM to delete the task
        task = Task.objects.get(id = pk)
        task.delete()
        # redirects user to index page
        return HttpResponseRedirect(reverse(f'tasktracker:index'))

    else:
        return HttpResponseRedirect(reverse(f'login'))
```

  #3:Incremental ID

```
# fix so that the id is not auto created (set to false now) as it was being auto incremeneted, need randomness for security
('id', models.AutoField(auto_created=False, primary_key=True, serialize=False, verbose_name='ID')),
```

  #4: Security Misconfiguration

```
# SECURITY WARNING: don't run with debug turned on in production!
# FIXED - turned debug to false and added host to allow
DEBUG = False

ALLOWED_HOSTS = ['127.0.0.1']
```

#5: Security Key
I set the key in my os environment based on these instructions. I could not get this to run without error and am unsure how it would work without a similar setup on another machine.

The other option to set the key in another file did not seem safe either. Couldn't the attacker access that file as well, it would be a bit more leg work I assume but still possible.

```
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = os.environ['SECRET_KEY']
#original security key: '#n6kks0cs$a-7k*67)k(nof$7z6&l+u97ea)nl_r8frg_mrmd1'
```

- **Q2:** Include in the code comments an explanation about the fix.

🤔 Hints/FAQs:

- Use the OWASP Top 10 list as a "checklist" for issues to look for in the code.
    - 4 vulnerabilities were intentionally added
- Some websites with tutorials/documentations that could help you fix certain issues:
    - Django ORM – Inserting, Updating & Deleting Data
    - Settings | Django documentation
    - Models | Django documentation
    - Django Tutorial Part 8: User authentication and permissions - Learn web development | MDN
    - Built-in template tags and filters | Django documentation

Grading Rubric

The HW will be evaluated based on the following criteria:

- **Comprehension**:
    - Does the answers correctly pinpoint the vulnerabilities?
- **Functionality (Produced output/answer)**:
    - Is the implemented solution fixing the identified vulnerabilities?
    - Is the code free from runtime errors?
- **Artifacts' Completeness & Quality**
    - Does the submission include all the required deliverables listed in this write-up?